Enabling Cross-Domain Communication: How to Bridge the Gap between AI and HW Engineers

Michael J. Klaiber, Axel J. Acosta, Ingo Feldner, Falk Rehm
firstname.surname@de.bosch.com
Robert Bosch Corporate Research, Renningen
Germany

ABSTRACT

A key issue in system design is the lack of communication between hardware, software and domain expert. Recent research work shows progress in automatic HW/SW co-design flows of neural accelerators that seems to make this kind of communication obsolete. Most real-world systems, however, are a composition of multiple processing units, communication networks and memories. A HW/SW co-design process of (reconfigurable) neural accelerators, therefore, is an important sub-problem towards a common co-design methodology. The ultimate challenge is to define the constraints for the design space exploration on system level - a task which requires deep knowledge and understanding of hardware architectures, mapping of workloads onto hardware and the application domain, e.g. artificial intelligence. For most projects, these skills are distributed among several people or even different teams which is one of the major reasons why there is no established end-to-end development methodology for digital systems. This position paper discusses possibilities how to establish such a methodology for systems that include (reconfigurable) dedicated accelerators and outlines the central role that languages and tools play in the process.

1 INTRODUCTION

Effective hardware/software co-design methodologies have been examined for decades and are still not the norm in the development of large-scale (embedded) systems [7, 10]. This often results from sequential or circular dependencies in the concept or requirements phase of a project. In particular, this means that hardware architects need workload specifications to dimension communication and computational resources, and application domain engineers, such as AI algorithm engineers, require information about the hardware before they can work efficiently. As this paper deals mostly with AI workloads, in particular neural networks, we use the term AI engineer synonymously with application domain engineer or software engineer to describe the person who transforms a specification or solution to a problem into executable code.

Many development methods such as the waterfall model, however, do not account for ramifications of algorithm to hardware and vice versa. Even if algorithm developers, software developers or hardware developers use agile development methods, there is no established communication medium to exchange requirements between their development paths. Assumptions in the different disciplines can't be exchanged easily. A result is that hardware is designed with assumptions about a workload which might already be outdated, and the algorithm is designed without an understanding what operations provide good performance on the target hardware.

This holds especially true in the field of embedded systems, as these systems often have low computing and communication resources and heterogenous project specific platforms.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
LATTE '21, April 15, 2021, Virtual, Earth
© 2021 Copyright held by the owner/author(s).

The methodology proposed in this paper should help to answer questions like:

- What implication does a specific algorithm change have on the system performance?
- What are the operations that are worth to be accelerated in hardware?
- Would a rearranging of physical memories or memory layout improve performance?
- How much more chip area and/or power do I want to spend for x% more accuracy or y% faster execution time?
- How to optimize an algorithm to perform best on a target platform without any hardware changes?

State-of-the-art methods show sophisticated methods for

- HW/SW co-design and generation of neural network accelerators [4, 9],
- Virtual Hardware Models [6] and
- Generation of Virtual Models from RTL Code [11].

The ideas presented in the following are mostly possible due to the recent progress in neural network compiler stacks, such as TVM [1] or Glow [8], used in combination with with virtual hardware and system models as bridging technology.

2 METHODOLOGY

2.1 The AI Engineer's Perspective

The primary objective of most AI Engineers is to design algorithmic models that achieve high prediction accuracy for a given data set. This model defines the workload to be executed on the target hardware.

In Fig. 1, the AI engineer's perspective is represented by:

- the **Algorithmic description**. This is an abstraction layer created by AI practitioners to formalize a problem with regards to algorithmic properties. Common formats are TensorFlow and ONNX.

The dominating technology that AI Engineers use for neural network training are GPUs and most likely server grade infrastructure. The execution time and memory footprint are, therefore, often secondary metrics that are evaluated based on the GPU's architectural properties and toolchains. This represent the first break of synergy of a AI/SW/HW team striving for joint development. An established workaround, is the use of cost functions to estimate low level implementations[3]. The difficulty arises in defining sophisticated and accurate cost functions for the target hardware. A task mostly handed over to the next stakeholders.

2.2 The Hardware Architect's Perspective

Hardware architects have many different tasks, among which particular important ones are to identify workload patterns that can be accelerated by dedicated computation blocks and to change the physical structure of the system to improve performance.

Workload patterns that occur often in a system can be accelerated by the use of special instructions on a CPU or by dedicated accelerators. Both possibilities require a detailed and formalized description of the workload to be executed. Whenever this workload description changes, considerations about the dedicated computation blocks can become obsolete.

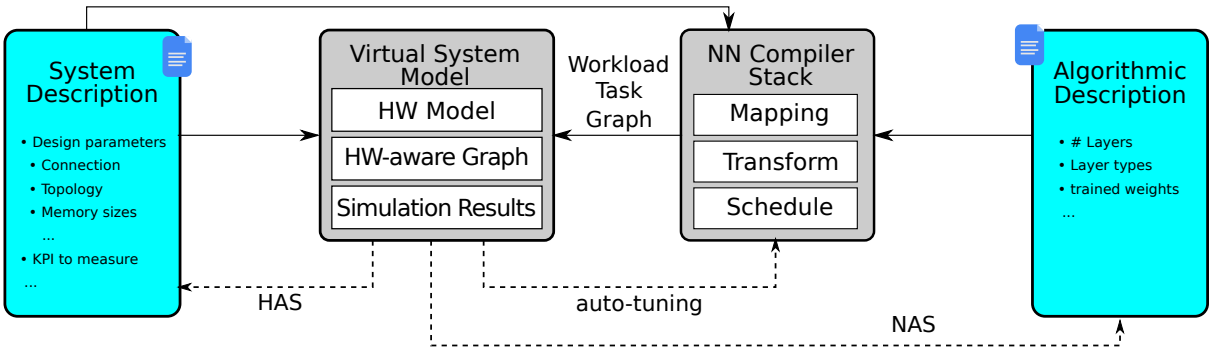

Figure 1: Development flow to enable system level cross-domain communication.

The physical connections, between computation blocks, communication networks and memories play a major role how fast a workload can be executed. Again, here a formalized workload description is the basis for decisions, e.g. to determine the size and granularity of the memories, how much bandwidth to allocate between a computation block and a memory, etc. A change of the workload description can make these considerations obsolete or inefficient.

In Figure 1, the HW architect's perspective is represented by:

- The **System description**: is an abstraction layer that provides a formalized description provided by the hardware architect. It contains the physical connections of hardware components, timing information of each hardware component requires to process an intrinsics value, and transaction behavior. For instance the time required to perform Conv2D instrinsic on an accelerator or the time to transfer 1 kByte of data from internal memory to the accelerator while five other hardware components request bus access. A common format in the automotive system context is Amalthea [2].
- The **Virtual hardware model**: is the executable form of the system description interfacing with the compiler. It essentially transforms the workload call graph into simulated execution time, memory accesses or other KPI.

HW development is a high effort and costly process. Therefore, there are few opportunities to make wide-reaching changes or adaptations after an architecture has been frozen. In order to cope with changes in the workload (or application) and enabling easier programmability, all computing HW systems must also offer a SW stack to automate the generation of implementations.

2.3 The Compiler Developer's Perspective

The compiler is the SW tool which bridges the abstraction between the AI engineer's domain and the HW architect's domain. It is given *the* daunting task of exploiting all HW properties designed into the architecture in order to create a good (or best) implementation of the AI engineer's model.

In Fig. 1, the compiler developer's perspective is represented by:

- The **Neural Network compiler stack**: The implementation of the model onto the hardware requires a mapping from AI engineer's domain to intrinsics of the (virtual) target hardware. For the mapping process it requires the intrinsics of all hardware components in the system. For optimization, the compiler needs timing and resource usage information of the hardware components. All of this information is contained in the system description. The transformation and optimization steps essentially formalizes a set of rules and knowledge that both hardware architect and AI engineers have about the system and the workload, and therefore, build the possibility for a fully automated exchange of information between the hardware architect and AI engineers.
- The **Workload task graph** is an abstraction layer consisting of a directed graph where each node represents an intrinsic call mapped to a hardware component of the system description. Each edge represents dependency relations. The task graph, for instances, describes a dependency like: the DMA transfer from extern memory to accelerator memory must be finished before computation on the transported data can be started.

2.4 From Isolated Optimizations to Holistic Solving

The dashed lines in Figure 1 show the paths for (automatic) optimization based on key performance indicators (KPI).

In isolation, each of the perspectives of the previous subsection has its own optimization problem and solutions:

- The AI engineer: optimizes the accuracy of a model via Network Architecture Search (NAS) methods[3].
- The HW architect: optimizes the HW resources via Hardware Architecture Search (HAS) methods [5].
- The Compiler developer: optimizes the implementation via auto-tuning methods[1]

The methodology we sketch in this paper tries to fill the gap that results from a lack of human communication in medium to large teams. However, the long-term goal of the community should be to extend this idea to combine all the optimization problems into a single one.

The conceptual break between these three optimization domains, would lend itself to a natural division of labor which leads to domain silos driving 3 different tool development. This, in our view, is a considerable organizational hurdle which will prevent effective communication between the teams. A solution to this, we argue, is to see only the one problem: "Creating a complete HW/SW stack for your product". Such a holistic view should also come with a more heterogeneous organization of teams, for example have a number of HW experts working on the higher levels of the stack (and vice versa). It is in such a setting that HW/SW/System co-design will become mainstream in large embedded system design.

3 CONCLUSION

HW/SW co-design for (reconfigurable) accelerators needs to take other components of the system into account. Compilers and languages are the key element to create a methodology that bridges the gap between AI and hardware engineers. In this paper we sketched a possible methodology and its components based on (mostly) existing technologies and show a new way how to bring optimization to the system level.

REFERENCES

[1] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Haichen Shen, Eddie Q Yan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: end-to-end optimization stack for deep learning. *arXiv preprint arXiv:1802.04799* 11 (2018), 20.

[2] AMALTHEA4public consortium. 2017. APP4MC help documentation.

[3] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. 2019. Neural architecture search: A survey. *J. Mach. Learn. Res.* 20, 55 (2019), 1–21.

[4] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. 2019. Fpga/dnn co-design: An efficient design methodology for 1ot intelligence on the edge. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[5] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. 2020. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 4154–4165.

[6] Michael J Klaiber, Sebastian Vogel, Axel Acosta, Robert Korn, Leonardo Ecco, Kristine Back, Andre Guntoro, and Ingo Feldner. 2019. An End-to-End HW/SW Co-Design Methodology to Design Efficient Deep Neural Network Systems using Virtual Models. In *Proceedings of the INTelligent Embedded Systems Architectures and Applications Workshop 2019*. 18–22.

[7] Glaydson Luiz Bertoze Lima, Guilherme Augusto Lopes Ferreira, Osamu Saotome, Adilson Marques Da Cunha, and Luiz Alberto Vieira Dias. 2015. Hardware development: Agile and co-design. In *2015 12th International Conference on Information Technology-New Generations*. IEEE, 784–787.

[8] Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Garret Catron, Summer Deng, Roman Dzhabarov, Nick Gibson, James Hegeman, Meghan Lele, Roman Levenstein, et al. 2018. Glow: Graph lowering compiler techniques for neural networks. *arXiv preprint arXiv:1805.00907* (2018).

[9] Zhan Shi, Chirag Sakhuja, Milad Hashemi, Kevin Swersky, and Calvin Lin. 2020. Learned Hardware/Software Co-Design of Neural Accelerators. arXiv:2010.02075 [cs.LG]

[10] Frank Slomka, Matthias Dorfel, Ralf Munzenberger, and Richard Hofmann. 2000. Hardware/software codesign and rapid prototyping of embedded systems. *IEEE Design & Test of Computers* 17, 2 (2000), 28–38.

[11] Wilson Snyder. 2017. Verilator: Speedy reference models, direct from RTL. *Presentation to University of Massachusetts Amherst* (2017).